

# SOAP vs REST: Comparing a master-slave GA implementation

P.A. Castillo

J.L. Bernier

M.G. Arenas

J.J. Merelo

P. García-Sánchez

**Abstract**—In this paper, a high-level comparison of both SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) is made. These are the two main approaches for interfacing to the web with web services. Both approaches are different and present some advantages and disadvantages for interfacing to web services: SOAP is conceptually more difficult (has a steeper learning curve) and more "heavy-weight" than REST, although it lacks of standards support for security. In order to test their efficiency (in time), two experiments have been performed using both technologies: a client-server model implementation and a master-slave based genetic algorithm (GA). The results obtained show clear differences in time between SOAP and REST implementations. Although both techniques are suitable for developing parallel systems, SOAP is heavier than REST, mainly due to the verbosity of SOAP communications (XML increases the time taken to parse the messages).

## I. INTRODUCTION

Service Oriented Architecture (SOA) [1] is a paradigm for organizing and utilizing distributed computational resources, called services. Using this paradigm, the service providers publish the descriptions (or interfaces) of the services they offer in a service registry, so the service requesters can discover them and bind to the correspondent service provider. The Web Services are the key point of integration for different applications belonging to different platforms, languages, systems since they are based in a set of standards that make them independent of the underlying technologies used for providing them.

Although there are several technologies for developing web services (SOAP, REST or XMLRPC among others [2], [3]), nowadays the main approaches are SOAP (Simple Object Access Protocol) [4], [5] and REST (Representational State Transfer) [6]. Both implementations are suitable for designing Web Services, however, it is important to understand the pros and cons of each one.

SOAP is the traditional, standards-based approach, but the majority of the web services with public API offer REST interfaces, while some of them offer both REST and SOAP and very few offer just SOAP. All of the major Web Services providers use REST: Twitter, Yahoo's, Flickr, delicio.us, pubsub, bloglines, technorati, and several others. Both eBay and Amazon have Web Services for both REST and SOAP.

On the other hand, SOAP Web Services are used in lots of enterprise software as well; for example, Google implements their Web Services using SOAP, with the exception of Blogger, which uses XML-RPC, an early and simpler pre-standard of SOAP.

The philosophies of SOAP and RESTful Web Services are very different. Strictly, SOAP is a protocol for distributed computing, whereas REST adheres much more closely to a web-based design. SOAP requires a greater implementation and understanding effort of the client side to difference of REST based APIs, that focus these efforts on the server side. Table I shows the main strengths and weaknesses for both SOAP and REST.

It is important to note that one of the advantages of SOAP is the use of the "generic" transport. While REST today uses HTTP/HTTPS, SOAP can use almost any transport to send the request. However, one perceived disadvantage is the use of XML because of its verbosity, and the time necessary to parse it.

In this way, in order to determine the efficiency of these two interfacing approaches, we have performed two experiments in which both a SOAP and REST implementations are evaluated:

- **Experiment 1:** a client-server model is implemented, in which the server process runs on a machine and the client processes send and receive text strings.
- **Experiment 2:** a master-slave based GA is implemented, running on the master process the GA and the fitness evaluation on the slave processes.

This work continues with our previous research in service oriented algorithms, as previously stated in [7], where a service-oriented platform was presented, or [8], where studies about P2P distributed evolutionary algorithms were performed.

This paper is structured as follows: In sections II and III a comprehensive description of SOAP and REST technologies are provided, respectively. Section IV describes the experiments. In concrete, the client-server and master-slave models implemented for testing are described, so the experimental configuration, the methodology considered in the study ; finally, the results obtained are shown. Last section (Section V), throws some conclusions and presents the proposed future work.

## II. SOAP: SIMPLE OBJECT ACCESS PROTOCOL

SOAP is a standard protocol proposed by the W3C ([4], [5]) to interface Web Services, and that extends the remote procedure call (XML-RPC). Thus, SOAP can be considered as an evolution of XML-RPC protocol, much more complete and mature, that allows to perform remote procedure calls to distributed routines (services) based on an XML interface as interfacing language. Thus, SOAP clients can access to objects and methods that are residing in remote servers, using an standard mechanism that makes transparent the details of implementation, such as the programming language of the

TABLE I  
STRENGTHS AND WEAKNESSES FOR BOTH SOAP (ABOVE) AND REST (BELOW).

SOAP	
Strengths (pros)	Weaknesses (cons)
<ul style="list-style-type: none"> <li>+ Handle distributed computing environments</li> <li>+ Built-in error handling</li> <li>+ Extensibility</li> <li>+ Language, platform, and transport agnostic</li> <li>+ Prevailing standard for web services</li> <li>+ Support from other standards (WSDL, WS-*)</li> </ul>	<ul style="list-style-type: none"> <li>- More verbose</li> <li>- Harder to develop, requires tools</li> <li>- Conceptually more difficult, more "heavy-weight" than REST</li> </ul>
REST	
Strengths (pros)	Weaknesses (cons)
<ul style="list-style-type: none"> <li>+ Language and platform agnostic</li> <li>+ Much simpler to develop than SOAP</li> <li>+ Small learning curve, less reliance on tools</li> <li>+ Concise, no need for additional messaging layer</li> <li>+ Closer in design and philosophy to the Web</li> </ul>	<ul style="list-style-type: none"> <li>- Assumes a point-to-point communication model</li> <li>- Not usable for distributed computing environment</li> <li>- Lack of standards support for security, etc.</li> <li>- Tied to the HTTP transport model</li> </ul>

routines, the operating system or the platform used by the provider of the service. At the moment, there exist complete implementations of SOAP for Perl, Java, Python, C++ and other languages [9]. In opposite to other remote procedure call methods, such as RMI (*remote method invocation*, used by the Java language) or XML-RPC, SOAP has two main advantages: it can be used with any programming language, and it can use any type of transport (HTTP, SHTTP, TCP, SMTP, POP and other protocols).

SOAP sends and receives messages using XML [10], [11], [12], wrapped HTTP-in headings. The interfaces of the methods that can be accessed using SOAP services are specified by a Web Services Description Language (WSDL) [13], [14]. The WSDL of an Web Service consists in an XML description of its interface, i.e., it is a file that describes the name of the methods, the parameters and type of data, the type of response that the Web Service may return, etc. Using an WSDL file, that it is based on a neutral language such as XML, the service can be specified for different languages, so that a Java client can access a Perl server.

In this way, SOAP constitutes a high level protocol, making easy the task of distributing objects among different servers, and avoiding the difficulties derived of defining the message formats, nor the explicit call to remote servers.

### III. REST: REPRESENTATIONAL STATE TRANSFER

After some years, Internet architects have found an alternative method for building web services in the form of Representational State Transfer (REST) [6].

REST is a style of software architecture for distributed hypermedia systems such as the World Wide Web. The term Representational State Transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation [15], [16]. Fielding is one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification versions 1.0 and 1.1 [17], [18].

REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed.

Although REST was initially described in the context of HTTP, is not limited to that protocol. RESTful architectures can be based on other Application Layer protocols if they already provide a rich and uniform vocabulary for applications based on the transfer of meaningful representational state. RESTful applications maximize the use of the pre-existing, well-defined interface and other built-in capabilities provided by the chosen network protocol, and minimize the addition of new application-specific features on top of it.

In a REST environment, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface is not altered. Finally, servers are able to temporarily extend or customize the functionality of a client by transferring logic to it that it can execute.

### IV. SOAP VS REST: COMPARING EFFICIENCY

In this paper we carry out two experiments to compare two parallel models implemented using SOAP and REST technologies in Perl language (due to the familiarity of the authors with this language [19], [20], [21]).

The SOAP model was implemented using the SOAP::Lite<sup>1</sup> [22] module, while the REST implementation was carried out using the Perl Dancer<sup>2</sup> module [23], [24], for their stability. In addition, servers developed using these

<sup>1</sup><http://www.soaplite.com>

<sup>2</sup><http://perldancer.org>

modules are easy to implement and deployed using the computer infrastructure available to us in our department.

The Experiment 1 consisted in the implementation of a client-server model. In this case, the server process runs on a machine that attends client requests, involving different lengths of text strings. The experiment 2 implements a master-slave based GA. In this case, a master process runs the GA, while different slave processes evaluate the fitness function.

#### A. Proof of Concept: Client-Server Efficiency Comparison

A classic client-server model is implemented in which clients can send and receive a text string. Different string lengths (100 and 1000 chars) have been configured in order to probe with different loads. In this way, we have tried to determine how the string length (the amount of data) affects the running time (due to communications).

Figures 1 and 2 show the Perl source code of the client-server SOAP and REST implementations.

The implementation of this experiment was conducted running the server process on a Ubuntu/Linux machine, while the clients were run on a Windows 7 with the Cygwin<sup>3</sup> environment.

As string lengths, values of 100 and 1000 characters have been used, in order to test whether the communications time depends on the amount of information sent. In both cases, the experiment was repeated for 50 times measuring the time spent using "gettimeofday" function (in order to achieve a good precision).

As shown in Table II, the SOAP version takes more time to complete the communications than the REST implementation.

TABLE II

RESULTS OBTAINED ON THE FIRST EXPERIMENT (CLIENT-SERVER IMPLEMENTATIONS). SOAP VERSION TAKES A SLIGHTLY HIGHER TIME, ALTHOUGH DIFFERENCES BETWEEN SENDING A 100 CHARS STRING AND A 1000 CHARS STRING ARE SMALLER.

	sending 100 chars	sending 1000 chars
SOAP	5.64 ± 0.17	5.83 ± 0.17
REST	2.56 ± 0.10	3.45 ± 0.10

SOAP version takes a slightly higher time, although differences between sending a 100 chars string and a 1000 chars string are smaller. REST implementation is faster due to the fact that no extra XML information is sent (that reduces the time taken to parse the messages).

#### B. Master-Slave based GA Implementation

In the Experiment 2, we have parallelized a GA following a master-slave model. We do not intend to innovate in terms of the parallel model, but in the implementation (because implementation matters [21]).

<sup>3</sup><http://www.cygwin.com>

There are many ways to implement a distributed genetic algorithm, one of which is the global parallelization (*farming*), in which, as Fogarty and Huang propose [25], Abramson and Abela [26], or Hauser and Männer [27], individual evaluation and/or genetic operator application are parallelized. A master processor supervises the population and select individuals to mate; then slave processors receive the individuals to evaluate them and to apply genetic operators.

An ideal client-server implementation of a distributed evolutionary algorithm could be a server process with several threads. Each thread would include a population, and would communicate with other threads through the shared code among them. Each thread would use an own tail of individuals to send to other threads. Each thread would evaluate its individuals in different remote computers, carrying out the communication using a REST server.

However, as we cannot use a threaded version of the Perl modules, our implementation will focus on the fitness function evaluation.

Thus, the simplest way of task distribution along this model is to evaluate the individual fitness function on the clients and to do the other steps on a master process (as shown in Figure 3); this scheme is usually called *farming*.

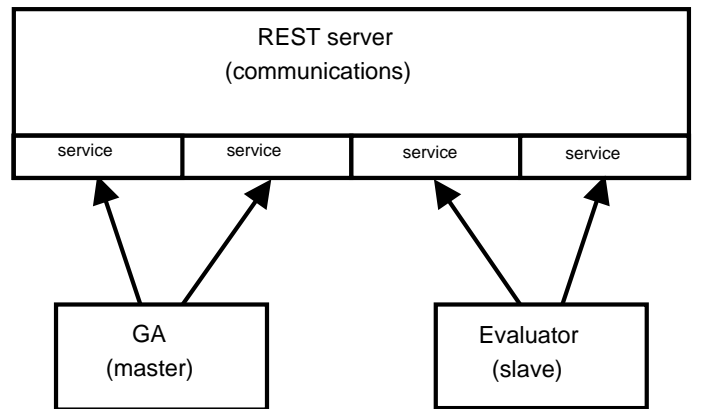


Fig. 3

SCHEMA OF THE MASTER-SLAVE BASED GA IMPLEMENTED IN THE SECOND EXPERIMENT. THE MASTER PROCESS RUNS THE GA AND THE SLAVE PROCESSES EVALUATE THE FITNESS FUNCTION.

The evolutionary algorithm has been implemented using the Algorithm::Evolutionary (A::E) library [19], [28]. Version 0.76.2 is used in this work, available at <http://opear.sourceforge.net> under GPL license.

In this experiment, the fitness function is devoted to optimize the function given by equation 1, which is plotted in Figure 4. Our aim is to find the optimum ( $f(0,0) = 1$ ) with an accuracy of  $10^{-6}$ .

$$f(x, y) = 1 + \frac{\sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}} \quad (1)$$

<pre> use SOAP::Transport::HTTP; my \$daemon = SOAP::Transport::HTTP::Daemon     -&gt; new (LocalPort =&gt; 80)     -&gt; dispatch_to('Demo'); \$daemon-&gt;handle;  package Demo; our \$src=""; sub push {     my (\$class, \$cad) = @_;     \$self-&gt;src = \$cad;     return "ok"; }; sub pop {     my \$tmp = \$self-&gt;src;     \$self-&gt;src = " ";     return \$tmp; }; </pre>	<pre> use Time::HiRes qw( gettimeofday tv_interval); use SOAP::Lite; my \$i=0; my \$tmp_it = [gettimeofday()]; for (\$i=0; \$i&lt;100 ; \$i++) {     my \$cad="01234567890 ... 01234567890";     print SOAP::Lite         -&gt; uri('http://www.soaplite.com/Demo')         -&gt; proxy('http://vaio/')         -&gt; push(\$cad) -&gt; result;     print SOAP::Lite         -&gt; uri('http://www.soaplite.com/Demo')         -&gt; proxy('http://vaio/')         -&gt; pop() -&gt; result; }; print "TIME: ", tv_interval( \$tmp_it ); </pre>
--	---

Fig. 1

SOAP PROGRAMMING EXAMPLE: SERVER (LEFT) AND CLIENT (RIGHT). THE STRING \$CAD VALUE VARIES FROM 100 TO 1000 CHARS IN ORDER TO CONFIGURE DIFFERENT LOADS.

<pre> use Dancer; my \$src = ""; get '/pop/' =&gt; sub {     my \$tmp = \$src;     \$src = " ";     return \$tmp; }; get '/push/:cad' =&gt; sub {     my (\$class, \$cad) = @_;     \$src = \$cad;     return "ok"; }; Dancer-&gt;dance; </pre>	<pre> use Time::HiRes qw( gettimeofday tv_interval); use LWP; my \$nav = new LWP::UserAgent; \$nav-&gt;agent("RESTzilla"); my \$i=0; my \$tmp_it = [gettimeofday()]; for (\$i=0; \$i&lt;100 ; \$i++) {     my \$cad="01234567890 ... 01234567890";     my \$rpush = new HTTP::Request GET         =&gt; 'http://127.0.0.1:3000/push/'."\$cad";     my \$upush = \$nav-&gt;request(\$rpush);     my \$rpop = new HTTP::Request GET         =&gt; 'http://127.0.0.1:3000/pop/';     my \$upop = \$nav-&gt;request(\$rpop); }; print "TIME: ", tv_interval( \$tmp_it ); </pre>
---	---

Fig. 2

REST PROGRAMMING EXAMPLE: SERVER (LEFT) AND CLIENT (RIGHT). THE STRING \$CAD VALUE VARIES FROM 100 TO 1000 CHARS IN ORDER TO CONFIGURE DIFFERENT LOADS.

GA individuals are represented using bitstrings (data type `A::E::Individual::bitstring`). As genetic operators, a bitflip mutation (`A::E::Op::Mutation`) and a two points crossover (`A::E::Op::Crossover`) are used.

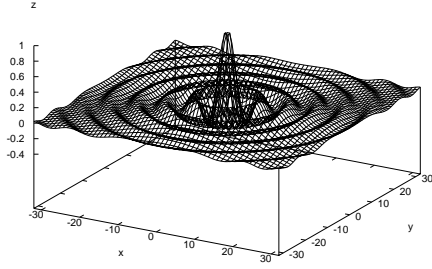


Fig. 4

FITNESS FUNCTION REPRESENTATION (GIVEN BY EQUATION 1). THE OPTIMUM OF THIS FUNCTION IS  $f(0, 0) = 1$ .

Remainder GA parameter values are set as follows (default values are used, since we do not intend to find the optimal ones, but to prove feasibility of the implementation, and carry out a comparison):

- Population size = 50
- Generations = 20
- Mutation rate = 20%
- Crossover rate = 80%
- Selection rate = 40%

The full source code (servers, GA and evaluators) and experiment data are available under GPL at: <http://atc.ugr.es/pedro/RESTvsSOAP.tgz>

As seen in Table III, the REST implementation is faster, due to the SOAP verbosity and time taken to decode the XML messages.

TABLE III

RESULTS OBTAINED ON THE SECOND EXPERIMENT (MASTER-SLAVE IMPLEMENTATIONS). BOTH IMPLEMENTATIONS OBTAIN GOOD RESULTS USING EVEN A SMALL NUMBER OF GENERATIONS AND POPULATION SIZE. AS FAR AS THE RUNNING TIME IS CONCERNED, REST IMPLEMENTATION IS FASTER IN BOTH CONFIGURATIONS (10 GEN. / 10 INDIV. AND 20 GEN. / 50 INDIV.).

		10 generations 10 individuals	20 generations 50 individuals
SOAP	accuracy	$0.997942 \pm 0.000762$	$0.999867 \pm 0.000101$
	time (sec.)	$3.79 \pm 0.42$	$31.03 \pm 1.89$
REST	accuracy	$0.996092 \pm 0.004081$	$0.999976 \pm 0.000003$
	time (sec.)	$2.06 \pm 0.08$	$15.05 \pm 1.17$

Both implementations obtain good results in terms of accuracy (both find the optimum with an accuracy of  $10^{-6}$ ) using even a small number of generations and population size. As far as the running time is concerned, REST implementation is faster for both load configurations. As in the client-server experiment, it might be due to the XML verbosity of SOAP communications (that increases the time taken to parse the messages).

## V. CONCLUSIONS

As reported in the experiments provided, both techniques are suitable for developing parallel systems. However, SOAP is heavier than REST, due to the verbosity of SOAP communications (XML increases the time taken to parse the messages).

On another hand, REST technology could not be used to implement a distributed GA following the island model as it does not support asynchronous processing and invocation, while SOAP does support it.

We can conclude that each technology approach has their uses. Moreover, they both have pros and cons. However we can devise some applications/situations where one of them might work better than the other:

- REST is more suitable if...
  - bandwidth and resources are limited
  - stateless *CRUD* (Create, Read, Update, and Delete) operations are needed (operation does not need to be continued)
  - the information can be cached because of the totally stateless operation of the REST approach
- SOAP is a good solution if...
  - the application needs asynchronous processing and invocation
  - the application needs a guaranteed level of reliability and security
  - both sides (provider and consumer) have to agree on the exchange format (rigid specifications)
  - the application needs contextual information and conversational state management (stateful operations)

From these REST implementations, several paths for improvement are devised: changing the models so that more computation is moved to the clients, leaving the server as just a hub for information interchange among clients; that information interchange will have to be reduced to the minimum. That will make this model closer to the island model, with just the migration policies regulated by the server. That way, the server bottleneck is almost eliminated.

As future research, it could be of interest adding support for SOAP and REST to existing distributed evolutionary algorithm libraries, such as JEO [29], EO [30], and libraries in other languages, in order to allow the implementation of multi-language evolutionary algorithms. In these experiments, Perl language has been used. It could be interesting to test these technologies using other programming languages and libraries (i.e. Java).

## ACKNOWLEDGEMENTS

This work has been supported in part by the CEI BioTIC GENIL (CEB09-0010) MICINN CEI Program (PYR-2010-13) project, the Junta de Andalucía TIC-3903 and P08-TIC-03928 projects, and the Jaén University UJA-08-16-30 project.

## REFERENCES

- [1] M. Papazoglou and W.-J. van den Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol. 16, pp. 389–415, 2007, 10.1007/s00778-007-0044-3.
- [2] Michaeldehaan, "XMLRPC vs REST vs SOAP vs all your RPC options," <http://bit.ly/MPRXA>, 2011.
- [3] oluyede, "If XMLRPC is really better than REST its not for there reasons," <http://bit.ly/bThgss>, 2011.
- [4] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000," Available from <http://www.w3.org/TR/SOAP>.
- [5] P. Ávila, "SOAP: revolución en la red," *Linux actual*, no. 19, pp. 55–59, 2001.
- [6] Wikipedia, "Representational State Transfer," [http://en.wikipedia.org/wiki/Representational\\_State\\_Transfer](http://en.wikipedia.org/wiki/Representational_State_Transfer), 2011.
- [7] P. García-Sánchez, J. González, P. A. Castillo, J. J. M. Guervós, A. M. Mora, J. L. J. Laredo, and M. G. Arenas, "A distributed service oriented framework for metaheuristics using a public standard," in *Nature Inspired Cooperative Strategies for Optimization, NISCO 2010, May 12-14, 2010, Granada, Spain. Studies in Computational Intelligence*, vol. 284, 2010, pp. 211–222.
- [8] J. L. J. Laredo, J. J. M. Guervós, and P. A. C. Valdivieso, "Evolvable agents: A framework for peer-to-peer evolutionary algorithms," in *Parallel and Distributed Computational Intelligence*, ser. Studies in Computational Intelligence, 2010, vol. 269, pp. 43–62.
- [9] soaprpc.com, "SOAP software," Available from <http://www.soaprpc.com/software>, 2011.
- [10] E. T. Ray, *Learning XML: creating self-describing data*. O'Reilly, January 2001.
- [11] E. R. Harold, *XML Bible*. IDG Books worldwide, 1991.
- [12] D. Box, "Inside SOAP," Available from <http://www.xml.com/pub/a/2000/02/09/feature/index.html>, 2011.
- [13] A. Ryman, "Understanding web services," Available from <http://www7.software.ibm.com/vad.nsf/Data/Document4362?OpenDocument&p=1&BCT=1&Footer=1>, 2011.
- [14] V. Vasudevan, "A web services primer," Available from <http://www.xml.com/pub/a/2001/04/04/webservices/index.html>, 2011.
- [15] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," *Doctoral dissertation, University of California, Irvine*, 2000.
- [16] R. Fielding and R. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology (TOIT) (New York: Association for Computing Machinery)* 2 (2): 115-150, 2002.
- [17] IETF, "RFC 1945," <http://tools.ietf.org/html/rfc1945>, 2011.
- [18] —, "RFC 2616," <http://tools.ietf.org/html/rfc2616>, 2011.
- [19] J.-J. Merelo, "A perl primer for evolutionary algorithm practitioners," *SIGEVolution*, vol. 4, no. 4, pp. 12–19, 2010.
- [20] J.-J. Merelo, A. Mora, P. Castillo, J.-L.-J. Laredo, and C. Fernandes, "Optimizing evolutionary algorithms at program level," in *Proceedings META 2010, International Conference on Metaheuristics and Nature Inspired Computing*, October 2010, proceedings available from <http://www2.lifl.fr/META10/pmwiki.php?n=Main.Proceedings>.
- [21] J. Merelo, G. Romero, M. Arenas, P. Castillo, A. Mora, and J. Laredo, "Implementation matters: programming best practices for evolutionary algorithms," in *To appear in Proceedings of IWANN2011*, 2011.
- [22] P. Kuchenko, "SOAP::Lite," Available from <http://www.soaplite.com>, 2011.
- [23] A. Sukrieh, "PerlDancer. The easiest way to write web applications with Perl," <http://perldancer.org>, 2011.
- [24] —, "PerlDancer. Documentation," <http://perldancer.org/documentation>, 2011.
- [25] T. Fogarty and R. Huang, "Implementing the genetic algorithm on transputer based parallel processing systems," *Parallel Problem Solving From Nature*, p. 145-149, 1991.
- [26] J. Abramson and A. Abela, "Parallel genetic algorithm for solving the school timetabling problem," in *Proceedings of the Fifteenth Australian Computer Science Conference (ACSC-15)*, vol. 14, p. 1-11, 1992.
- [27] R. Hauser and R. Männer, "Implementation of standard genetic algorithm on mimd machines," in *Davidor Y., Schwefel H. P., Männer R., Eds., Parallel Problem Solving from Nature, PPSN III*, p. 504-513, Springer-Verlag (Berlin), 1994.
- [28] J.-J. Merelo-Guervós, P.-A. Castillo, and E. Alba, "Algorithm::Evolutionary, a flexible Perl module for evolutionary computation," *Soft Computing. A Fusion of Foundations, Methodologies and Applications. Springer Berlin / Heidelberg. issn 1432-7643. Vol.14, issue 10, pp. 1091-1109*, 2010.
- [29] M. Arenas, L. Foucart, J.-J. Merelo-Guervós, and P. A. Castillo, "JEO: a framework for Evolving Objects in Java," in *Actas Jornadas de Paralelismo*, UPV. Universidad Politécnica de Valencia, 2001, pp. 185–191.
- [30] M. Keijzer, J.-J. Merelo-Guervós, G. Romero, and M. Schoenauer, "Evolving Objects: a general purpose evolutionary computation library," in *Artificial Evolution, 5th International Conference, Evolution Artificielle, EA 2001, Le Creusot, France, October 29-31, 2001, Selected Papers*, ser. Lecture Notes in Computer Science, P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, and M. Schoenauer, Eds., vol. 2310. Springer, 2002, pp. 231–244, <http://citeseer.nj.nec.com/context/2155872/0>, <http://www.springerlink.com/link.asp?id=qpc0fdxgt3523m4r>.